TleraBoards

Release 1.0

Luighi Viton

Sep 19, 2020

CONTENTS:

1 Initial setup **3**
- 1.1 1. Required software . . . 3
- 1.2 2. Install base libraries . . . 5

2 Library installation **7**
- 2.1 1. Get the repository . . . 7
- 2.2 2. Repository structure . . . 7
- 2.3 3. Loading the project in Visual Studio code . . . 8

3 Configuration files **11**
- 3.1 1. Sensor Configuration . . . 11
- 3.2 2. Boards declaration . . . 12
- 3.3 3. Radio communication settings (Commissioning) . . . 13

4 Sensors Tlera Documentation **15**
- 4.1 1. Type declarations . . . 15
- 4.2 2. Class declaration . . . 25

5 Application example **29**
- 5.1 1. Application structure . . . 29
- 5.2 2. Uploading application . . . 30

6 Building documentation **31**
- 6.1 1. Local building process . . . 31
- 6.2 2. Documentation at ReadTheDocs . . . 32
- 6.3 3. Additional resources . . . 32

7 About **33**
- 7.1 1. Project description . . . 33
- 7.2 2. Project team . . . 33

Index **35**

Welcome to the TleraBoards' documentation. This archive was created to hold the documentation related to Tlera boards used for radiocommunication using the LoRa protocol. These are low-cost prototyping ready platforms which contains several embedded sensors and facilitates the implementation of IoT solutions.

This library is centered on the Cricket board which has also a GPS incorporated to localize the device using this technology. Nevertheless, it could be also applied to other boards with some modifications.

The library and documentation presented here, was developed as part of a project for Geolocalization using LoRa communications to study the behavioral of Southamerican camels, developed by researchers at INICTEL-UNI. If you want to know more about this project, you could visit the *About* page.

CHAPTER

ONE

INITIAL SETUP

The library is based on the library developed by Kris Winer, who is the creator of Tlera Boards. This library is originally thought for the Arduino IDE, so the library created in this documentation is meant to use with this framework as well.

So, to use it it is required to install first the Arduino framework, and an appropriate editor to develop your application.

1.1 1. Required software

1.1.1 1.1 Arduino IDE

First of all the required application to compile is the Arduino framework which is installed with the Arduino IDE.

For this you could perfom the next steps:

For Windows

1. Visit the Arduino software download page and select the appropriate version according to your system properties.
2. Once downloaded install the software in your system, taking into account the path where you intall it and the preferences file.

For GNU/Linux

Depending on your GNU/Linux distribution there will be a package appropriate for your system.

In this case, we provide the installation steps for ArchLinux, which was the system used for the test:

1. Install the `arduino-avr-core` and the `arduino` package:

```
sudo pacman -S arduino arduino-avr-core
```

2. Add the user to the `uucp` and `lock` user groups, according to the Arduino Documentation:

```
sudo usermod -a -G uucp username
sudo usermod -a -G lock username
```

After these steps are accomplished, you should relogin your session to these changes takes effect.

1.1.2 1.2 Visual Studio Code

Although the Arduino IDE is enough to create an application and use the board, we recommend using a more featured IDE which also have linters and allows to backtrace the libraries.

We recommend for this purpose, Visual Studio Code, with a convenient Arduino extension.

Visual Studio Code installation

To perfom this goal, follow the next steps:

For Windows

1. Download the software from the Visual Studio Code page according with your system properties.
2. Install the software following the common flow for Windows software installation.

For Linux

Depending of the linux dristribution the commands may vary. Fortunately, there are `.deb` and `.rpm` versions for the most common GNU/Linux distributions such as Debian derived ones (Ubuntu, and their flavours) and Red Hat derived ones (Fedora, CentOS, and others).

In case of ArchLinux, the command to install it is:

```
sudo pacman -S code
```

For more detils, could visit the Archlinux package documentation for Visual Studio Code.

Arduino extension installation

The recommended extension to work with Arduino in Visual Studio Code is Arduino. This extension could be installed from the extensions menu in the Code IDE.

To accomplish this goal you could perform the following steps:

1. Open the extensions menu in the side bar and search for `vscode-arduino`.
2. Select the extension provided by Microsoft.
3. Once openned install the extension by clicking on the Install green button.

It is probably that the extension when is installed suggest the C/C++ extension as the code for Arduino is primarily in this language. If the pop-up message doesn't appear we suggest installing it manually, by:

1. In the extensions menu, search for `c++`.
2. Select the extension provided by Microsoft.
3. Once openned install the extension by clicking on the Install green button.

For more information, could visit the extension repository.

1.2 2. Install base libraries

Normally, the Arduino extension grabs its configuration from the Arduino IDE preferences, so if you had installed some libraries, they will be available in the Arduino extension for Visual Code.

In this case, we want to install the library employing the preferences file from the Arduino extension.

The base library, according to the Tlera Board creator is the Arduino Core for STM32L0 based boards which must be installed to be able to compile the code in the application.

So, to install it, follow the next steps:

1. Open the user preferences for Arduino extension via `Ctrl + ,`.
2. Locate the ArduinoConfiguration.
3. Click on *Edit in settings.json*.
4. Add the following url *https://grumpyoldpizza.github.io/ArduinoCore-stm32l0/package_stm32l0_boards_index.json* to the `arduino.additionalUrls` option, resulting as follows:

```
"arduino.additionalUrls": [
   "https://grumpyoldpizza.github.io/ArduinoCore-stm32l0/package_stm32l0_
↪boards_index.json",
]
```

5. Save the file with `Ctrl + S`.

Now you can install the library core for this board via:

1. Open the Arduino command pressing `F1`.
2. Select `Arduino:Board Manager`.
3. Search for "Tlera Corp STM32L0 Boards"
4. Click on Install button.

Once the boards are installed you could close the Board Manager.

CHAPTER

TWO

LIBRARY INSTALLATION

To install the library you could clone the main repository where you can find the project and an application example.

2.1 1. Get the repository

To clone the repository you can follow the next steps:

```
git clone https://gitlab.com/projectlorawan-geolocalization/tleraboards/
cd tleraboards
```

2.2 2. Repository structure

The repository structure is as follows:

```
tleraboards/                    (main repository)
|--- .vscode/                   (visual code configuration)
|   |--- arduino.json
|   |--- c_cpp_properties.json
|   |--- settings.json
|--- cmake/                     (cmake module addition for sphinx)
|   |--- FindSphinx.cmake
|--- CMakeLists.txt             (main CMakeLists.txt file)
|--- docs/                      (documentation configuration folder)
|   |--- CMakeLists.txt         (cmake configuration for documentation)
|   |--- conf.py                (configuration for Sphinx)
|   |--- contents               (individual contents files)
|   |   |--- about.rst
|   |   |--- initialsteps.rst
|   |   |--- installationlibrary.rst
|   |--- Doxyfile.in            (Doxygen configuration)
|   |--- index.rst              (main documentation file)
|   |--- requirements.txt       (requirements for ReadTheDocs)
|--- LoRaDev.code-workspace
|--- src/                       (source code folder)
    |--- demo_cricket_lorawan_asset_tracker
       |--- ADXL345.cpp
       |--- ADXL345.h
       |--- BMA400.cpp
       |--- BMA400.h
       |--- BME280.cpp
```

(continues on next page)

(continued from previous page)

```
        |--- BME280.h
        |--- BMP085.cpp
        |--- BMP085.h
        |--- boards.h
        |--- CMakeLists.txt      (cmake configuration for source)
        |--- commissioning.h
        |--- demo_cricket_lorawan_asset_tracker.ino (application example)
        |--- HMC5883L.cpp
        |--- HMC5883L.h
        |--- I2Cdev.cpp
        |--- I2Cdev.h
        |--- L3G4200D.cpp
        |--- L3G4200D.h
        |--- libraries
        |    |--- readme.txt
        |--- sensor_config.h
        |--- SensorsTlera.cpp    (main class source)
        |--- SensorsTlera.h      (main class header)
        |--- SPIFlash.cpp
        |--- SPIFlash.h
```

There are some important folders and files to take into account:

.vscode This folder contains the configuration for the Visual Studio code These configurations are loaded when is opened with the IDE.

cmake The documentation uses cmake to generate it locally, so in this folder there is a custom module which integrates sphinx with cmake.

docs Is the documentation folder which contains all the files related to the documentation configuaration such as Doxygen and Sphinx files.

src Contains the source code of the project, which includes the libraries and the application example usign them.

2.3 3. Loading the project in Visual Studio code

The project is prepared to be loaded by Visual Studio Code, so to accomplish this goal, you should follow the next steps:

1. Open the Visual Studio Code.
2. Go to File-> Open workspace
3. In the file explorer, localize the `LoRaDev.code-workspace` file in the repository and Open it.

Once finished the last step, the whole project should be loaded into the IDE.

Then you can explore the following in the repository:

configuration files These files are composed by `commisioning.h`, `boards.h` y `sensor_config.h`, which provides the configuration for the library and are discused in *Configuration files*

main library The main library is composed by `SensorsTlera.h` and `SensorsTlera.cpp` which is explained in *Sensors Tlera Documentation*

application The application example is the Arduino file which employs the library and other sensors considered here. It is explained in *Application example*.

documentation The documentation files wich are used to build this web documentation and uses Sphinx and Doxygen, explored in *Building documentation*.

CHAPTER

THREE

CONFIGURATION FILES

The library is centered on the SensorsTlera class, which is described in `SensorsTlera.h` and `SensorsTlera.cpp` whose documentation is discussed in *Sensors Tlera Documentation*.

However, to use properly this library is required to create some configuration files which are explained here.

3.1 1. Sensor Configuration

These configurations are provided by `sensors_config.h` file.

There is two global definitions to take into account:

BOARD_TYPE
: Board configuration.

DEBUG
: Debug settings.

Then there are some groups of definitions which could be modified according to the required, but only the first two groups are suggested to be edited by user.

3.1.1 1.1. Sensors definitions

These definitions are related to the sensors which could be enabled or disabled in function of the corresponding definition is uncommented or commented.

WITH_GNSS
: Enable GNSS.

WITH_BMA400
: Enable BMA400 sensor.

WITH_BME280
: Enable BME280 sensor.

WITH_ADXL345
: Enable ADXL345 sensor.

WITH_L3G4200D
: Enable L3G4200D sensor.

WITH_HMC5883L
: Enable HMC5883L sensor.

3.1.2 1.2. Debug definitions

These definitions where created for debugging purposes, enabling them by assigning them a 1 value.

DEBUG_BOARD
Enable debugging for onboard sensors.

DEBUG_GNSS
Enable debugging for GNSS sensor.

DEBUG_BMA400
Enable debugging for BMA400 sensor.

DEBUG_BME280
Enable debugging for BME280 sensor.

DEBUG_ADXL345
Enable debugging for ADXL345 sensor.

DEBUG_L3G4200D
Enable debugging for L3G4200D sensor.

DEBUG_HMC5883L
Enable debugging for HMC5883L sensor.

3.1.3 1.3. Board definitions

These definitions varies according the `BOARD_TYPE`. In this case, those are set for the Cricket board.

LED_BOARD
LED blue.

VBAT_CTRL
Enable battery voltage monitoring.

VBAT_MON
ADC for Lipo voltage monitoring.

USER_BUTTON
user button

3.2 2. Boards declaration

These definitions are meant to assign an ID for each BOARD_TYPE.

For this purpose there are some macros and enumeration type declaration to generate an ID for each board type.

enum board_t
Board type enumeration

Values:

enumerator CRICKET
CRICKET type definition.

enumerator GNAT
GNAT type definition.

enumerator GRASSHOPPER
GRASSHOPPER type definition.

CRI (*x*)
CRICKET board macro generation.

GNAT (*x*)
GNAT board macro generation.

GRASS (*x*)
GRASSHOPPER board macro generation.

These macros are used to generate an ID for each board used to upload the application. For example, you could employ `CRI(x)` to generate an ID for a `x` board number:

```
CRI(4)  = 1004 // ID for the fourth Cricket board
```

In general, you shouldn't require to modify this file, unless you add other boards definitions.

3.3 3. Radio communication settings (Commissioning)

These settings are meant to configure the GATEWAY TYPE and the LoRaWAN parametters.

3.3.1 3.1. Gateway definitions

GATEWAY_TEKTELIC
Defines the gateway as TEKTELIC type.

For the network sever there are two options `GATEWAY_TEKTELIC` for Tektelic network server and `GATEWAY_TTN` for TTN network server.

3.3.2 3.2. LoRaWAN settings

In LoRaWAN settings, there are some which sould be modified according with personal settings of application server configurations such as:

MYAPPKEY
APPKey from network server.

Then, it is required to define the specific device to be flashed through:

MYDEVICE
Device identification.

In addition to this, there is a struct definition which holds all LoRaWAN settings:

struct LoRaSettings
Structure for storing *LoRaSettings*.

Public Members

const int **id**
Device ID.

const char ***appEui**
AppEui from application server.

const char ***appKey**
AppKey from application server.

const char ***devEui**
DevEui set in applicatio and device.

In fact, you do not require to create the structure manually. There is a macro which facilitates its creation:

LORA_SETTINGS(...)
Macro to create LoRaWAN settings using *LoRaSettings* struct.

LORA_SETTINGS_CRI(*NUM*, ...)
Macro to create LoRaWAN settings using *LoRaSettings* struct for Cricket devices.

You could use either `LORA_SETTINGS(...)` for a general definition or `LORA_SETTINGS_CRI(NUM, ...)` for a cricket definition.

On the other hand, there are several cricket definitions, which you could adapt to your own devices.

To sum, only you need to set the corresponding `MYAPPKEY` and the `MYDEVICE`, for example:

```
#define MYAPPKEY "XXXXXXX" // Your App key from application server
#define MYDEVICE CRI(2)    // The current device (in this case, 2)
```

In addition to this, you could add more definitions to allow other device settings as in the example file:

```
#if MYDEVICE == CRI(1)
LORA_SETTINGS_CRI(1, MYAPPKEY, "CRI-1-appkey", "CRI-1-deveui");
#elif MYDEVICE == CRI(2)
LORA_SETTINGS_CRI(2, MYAPPKEY, "CRI-2-appkey", "CRI-2-deveui");
#elif MYDEVICE == CRI(3)
LORA_SETTINGS_CRI(3, MYAPPKEY, "CRI-3-appkey", "CRI-3-deveui");
...
#elif MYDEVICE == CRI(n)
LORA_SETTINGS_CRI(n, MYAPPKEY, "CRI-n-appkey", "CRI-n-deveui");
#endif
```

CHAPTER

FOUR

SENSORS TLERA DOCUMENTATION

This is the main library of the project and is comoposed by two files:

SensorsTlera.h Header of the main class.

SensorsTlera.cpp Source file of the main class.

Those files describes a class which is `SensorsTlera` and declare some structs which are useful to manage data from sensors. Those declarations could be divided in two groups, the mangnitud structs and the sensor type structs which are explanined as follows.

4.1 1. Type declarations

4.1.1 1.1. Magnitude type declarations

These declarations are centered on the type of magnitude measured by sensors.

1.1.1. Acelerometer type

struct accel_t
Acceleration type.

Public Functions

operator uint8_t*()
Operator to convert into uint8_t array.

This operator save the result in the bytes array. It could be used to cast data to bytes.

The resulting array is:

```
    [ax1, ax2, ay1, ay2, az1, az2]
```

operator String()
Operator to convert into String type.

This operator returns a string type. Could be used to print data for debugging purposes.

The resulting string is:

```
  AX:axvalue;AY:ayvalue;AZ:azvalue
```

Public Members

int16_t **ax**
acceleration in x axis

int16_t **ay**
acceleration in y axis

int16_t **az**
acceleration in z axis

uint8_t **bytes**[6]
array of acceleration in bytes

1.1.2. Gyroscope type

struct gyro_t
Gyroscopo type.

Public Functions

operator uint8_t*()
Operator to convert into uint8_t array.

This operator save the result in the bytes array. It could be used to cast data to bytes.

The resulting array is:

```
[avx1, avx2, avy1, avy2, avz1, avz2]
```

operator String()
Operator to convert into String type.

This operator returns a string type. Could be used to print data for debugging purposes.

The resulting string is:

```
AVX:avxvalue;AVY:avyvalue;AVZ:avzvalue
```

Public Members

int16_t **avx**
angular velocity in x axis

int16_t **avy**
angular velocity in y axis

int16_t **avz**
angular velocity in z axis

uint8_t **bytes**[6]
array of angular velocity in bytes

1.1.3. Magnetometer type

struct mag_t
Magnetometer type.

Public Functions

operator uint8_t*()
Operator to convert into uint8_t array.

This operator save the result in the bytes array. It could be used to cast data to bytes.

The resulting array is:

```
[mx1, mx2, my1, my2, mz1, mz2]
```

operator String()
Operator to convert into String type.

This operator returns a string type. Could be used to print data for debugging purposes.

The resulting string is:

```
MX:mxvalue;MY:myvalue;MZ:mzvalue
```

Public Members

int16_t **mx**
magnetic field intensity in x axis

int16_t **my**
magnetic field intensity in y axis

int16_t **mz**
magnetic field intensity in z axis

uint8_t **bytes**[6]
array of magnetic field values

1.1.4. Temperature type

struct temp_t
Temperature type.

Public Functions

operator int16_t()
Operator to convert into uint16_t value.

operator uint8_t*()
Operator to convert into uint8_t array.

This operator save the result in the bytes array. It could be used to cast temperature to bytes.

The resulting array is:

```
[tempc1, tempc2]
```

operator String()
Operator to convert into String type.

This operator returns a string type. Could be used to print data for debugging purposes.

The resulting string is:

```
TEMPC:tempcvalue;TEMPF:tempfvalue
```

Public Members

int32_t **raw**
raw temperature value

int32_t **compensated**
compensated temperature value

float **temp_C**
temperature value in C degrees

float **temp_F**
temperature value in F degrees

uint8_t **bytes**[2]
byte array for temperature

1.1.5. Humidity type

struct hum_t
Humidity type.

Public Functions

operator int16_t()
Operator to convert into uint16_t value.

operator uint8_t*()
Operator to convert into uint8_t array.

This operator save the result in the bytes array. It could be used to cast humidity value to bytes.

The resulting array is:

```
[hum1, hum2]
```

operator String()
Operator to convert into String type.

This operator returns a string type. Could be used to print data for debugging purposes.

The resulting string is:

```
HUM:humidityvalue
```

Public Members

int32_t **raw**
Raw humidity value.

uint32_t **compensated**
Compensated humidity value.

float **hum_relative**
Relative humidity.

uint8_t **bytes**[2]
byte array for relative humidity value

1.1.6. Pressure type

struct press_t
Pressure type.

Public Functions

operator int16_t()
Operator to convert into uint16_t value.

operator uint8_t*()
Operator to convert into uint8_t array.

This operator save the result in the bytes array. It could be used to cast pressure value to bytes.

The resulting array is:

```
[altitude_mts1, altitude_mts2]
```

operator String()
Operator to convert into String type.

This operator returns a string type. Could be used to print data for debugging purposes.

The resulting string is:

```
PRESS:pressure_mbar;ALTITUDE:altitude_mts
```

Public Members

int32_t **raw**
Raw pressure value.

uint32_t **compensated**
Compensated pressure value.

float **press_mbar**
Pressure value in mbar.

float **altitude**
Altitude value.

float **altitude_mts**
Altitude value in meters.

uint8_t **bytes**[2]
Byte array to hold altitude.

1.1.7. Battery type

struct batt_t
Battery type.

Public Functions

operator String()
Operator to convert into String type.

This operator returns a string type. Could be used to print data for debugging purposes.

The resulting string is:

```
VBAT:vbatvalue
```

operator int16_t()
Operator to convert into uint16_t value.

operator uint8_t*()
Operator to convert into uint8_t array.

This operator save the result in the bytes array. It could be used to cast battery voltage to bytes.

The resulting array is:

```
[vbat1, vbat2]
```

Public Members

float **vdda**
Voltage value at VDDA.

float **vbus**
Voltage value at VBUS.

float **vbat**
Voltage value at VBAT.

float **STM32L0Temp**
Chip temperature.

uint8_t **bytes**[2]
Byte array for VBAT value.

4.1.2 1.2. Sensor type declarations

These declarations are oriented to hold the information by sensors.

1.2.1. GNSS type

struct GNSS_t
GNSS type.

Public Functions

operator uint8_t*()
Operator to convert into uint8_t array.

This operator save the result in the bytes array. It could be used to cast GPS information to bytes.

The resulting array is:

```
[altitude1, altitude2,
 latitude1, latitude2, latitude3, latitude4,
 longitude1, longitude2, longitude3, longitude4,
]
```

Where: altitude is in meters*10, altitude is float and longitude is float.

Public Members

bool **isSpatial** = false
Holds if it has spatial values.

GNSSLocation **location**
GNSSLocation object to manage location.

double **latitude**
Latitude value.

double **longitude**
Longitude value.

float **altitude**
Altitude value.

uint8_t **bytes**[10]
Byte array for GPS information.

1.2.2. BMA400 type

struct BMA400_t
BMA400 type.

Public Members

status_t **status**
sensor status

accel_t **values**
acceleration structure

float **resolution**
scale resolutions per LSB for the sensor

float **offset**[3]
accel bias offsets

1.2.3. BME280 type

struct BME280_t
BME280 type.

Public Functions

operator uint8_t*()
Operator to convert into uint8_t array.

This operator save the result in the bytes array. It could be used to cast sensor values to bytes.

The resulting array is:

```
[temperature1, temperature2,
 humidity1, humidity2,
]
```

operator String()
Operator to convert into String type.

This operator returns a string type. Could be used to print data for debugging purposes.

The resulting string is:

```
TEMP::temperature
HUM::humidity
PRESS::pressure
```

Public Members

status_t **status**
sensor status

temp_t **temperature**
temperature structure

hum_t **humidity**
humidity structure

press_t **pressure**
pressure structure

uint8_t **bytes**[4]
byte array to hold sensor values

1.2.4. ADXL345 type

struct ADXL345_t
ADXL345 type.

Public Functions

operator uint8_t*()
Operator to convert into uint8_t array.

This operator save the acceleration in the byte array.

The resulting array is:

```
[accel1, accel2, accel3, accel4, accel5, accel6]
```

operator String()
Operator to convert into String type.

This operator returns a string type. Could be used to print data for debugging purposes.

The resulting string is:

```
ACCEL::accelerationvalues
```

Public Members

status_t **status**
sensor status

accel_t **values**
acceleration structure

uint8_t **bytes**[6]
byte array to hold acceleration

1.2.5. L3G4200D type

struct L3G4200D_t
L3G4200D type.

Public Functions

operator uint8_t*()
Operator to convert into uint8_t array.

This operator save the gyroscope in the byte array.

The resulting array is:

```
[gyro1, gyro2, gyro3, gyro4, gyro5, gyro6]
```

operator String()
Operator to convert into String type.

This operator returns a string type. Could be used to print data for debugging purposes.

The resulting string is:

```
GYRO::gyroscopevalues
```

Public Members

status_t **status**
sensor status

gyro_t **values**
gyroscope structure

uint8_t **bytes**[6]
byte array to hold gyroscope value

1.2.6. HMC5883L type

struct HMC5883L_t
HMC5883L type.

Public Functions

operator uint8_t*()
Operator to convert into uint8_t array.

This operator save the magnetometer in the byte array.

The resulting array is:

```
[mag1, mag2, mag3, mag4, mag5, mag6]
```

operator String()
Operator to convert into String type.

This operator returns a string type. Could be used to print data for debugging purposes.

The resulting string is:

```
MAG::magnetometervalues
```

Public Members

status_t **status**
sensor status

mag_t **values**
magnetometer structure

float **heading**
heading value

uint8_t **bytes**[6]
byte array to hold magnetometer values

4.2 2. Class declaration

The main class is `SensorsTlera`, which is explored as follows:

class SensorsTlera
Class for Tlera Corp sensors.

Public Functions

SensorsTlera()
Constructor for Tlera Corp.

mcuid_t **getUID**()
Obtain UID from device.

Return Value

- `this->reg_ID`: return the uid

void **initialize_board**()
Initialize onboard elements.

void **initialize_GNSS**()
Initialize GNSS sensor.

Initialize the pins for GNSS, establishes the constellation, and the antenna type.

void **initialize_BMA400**()
Initialize BMA400 sensor.

void **initialize_BME280**()
Initialize BME280 sensor.

void **initialize_ADXL345**(ADXL345 *sensorADXL345*)
Initialize ADXL345 sensor.

Parameters

- `sensorADXL345`: object of ADXL345 sensor

void **initialize_L3G4200D**(L3G4200D *sensorL3G4200D*)
Initialize L3G4200D sensor.

Parameters

- `sensorL3G4200D`: object of L3G4200D sensor

void **initialize_HMC5883L**(HMC5883L *sensorHMC5883L*)
Initialize HMC5883L sensor.

Parameters

- `sensorHMC5883L`: object of HMC5883L sensor

void **callibrate_BMA400**()
Callibrate BMA400 sensor.

Reset sensor and self test, apply compensation parameters and initialize it.

void **callibrate_BME280**()
Callibrate BME280 sensor.

Reset sensor and initialize the sensor according to pre-defined values.

batt_t **readSensors_batmon**()
Obtain voltage values.

Obtain values from microcrontroller internal functions.

Return Value

- `this->battery_info`: Returns structure with battery information.

batt_t **readSensors_batmon_average**()
Obtain average voltage values.

Apply average and saves to internal battery_info structure.

Return Value

- `this->battery_info`: Returns structure with battery information.

BME280_t **readSensors_BME280**()
Read values from BME280 sensor.

Obtain value of temperature, humidity and pressure and save them to *BME280_t* structure.

Return Value

- `this->reg_BME280`: Returns structure with BME280 values.

ADXL345_t **readSensors_ADXL345**(ADXL345 *sensorADXL345*)
Read values from ADXL345 sensor.

Obtain value of acceleration and save them to *ADXL345_t* structure.

Parameters

- `sensorADXL345`: object of ADXL345 type

Return Value

- `this->reg_ADXL345`: Returns structure with ADXL345 values.

L3G4200D_t **readSensors_L3G4200D**(L3G4200D *sensorL3G4200D*)
Read values from L3G4200D sensor.

Obtain value of gyroscope and save them to *L3G4200D_t* structure.

Parameters

- `sensorL3G4200D`: object of L3G4200D type

Return Value

- `this->reg_L3G4200D`: Returns structure with L3G4200D values.

HMC5883L_t **readSensors_HMC5883L**(HMC5883L *sensorHMC5883L*)
Read values from HMC5883L sensor.

Obtain value of magnetometer and save them to *HMC5883L_t* structure.

Parameters

- `sensorHMC5883L`: object of HMC5883L type

Return Value

- `this->reg_HMC5883L`: Returns structure with HMC5883L values.

GNSS_t **readSensors_GNSS**()
Read values from GNSS sensor.

Obtain value from GPS and print according if It has spatial values.

Return Value

- `this->reg_GNSS`: Returns structure with GPS values.

bool **GNSS_isSpatial**()
Detects if GPS has spatial values.

void **scanI2CDevices**()
Scan I2C devices.

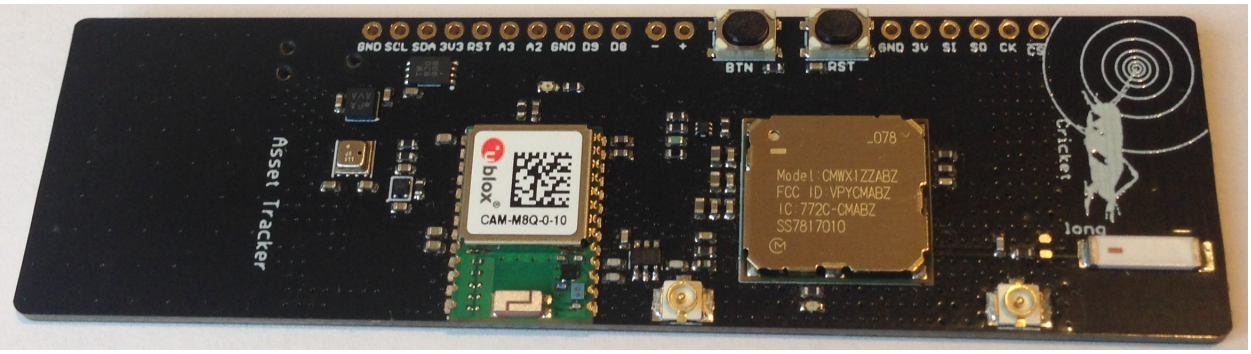
Method copied from BME280 library created by Kris Winer 06/16/2017 Copyright Tlera Corporation

CHAPTER

FIVE

APPLICATION EXAMPLE

The application is prepared for the Ckicket board, whose information could be find in the Hackaday page and the Tindie page where you can acquire it and it is used in the application employed in this project.



The main application is an Arduino File, an it has the structure and the compilation process according to the Arduino framework.

5.1 1. Application structure

As every Arduino application the program has two main parts which acquire `void setup()` and `void loop()`.

On the other hand, to use the library is requiered to create the object corresponding to the `SensorTlera` class.

An example usign the BME280 sensor, could have the following structure:

```
#include "SensorsTlera.h"
#include "sensor_config.h"

SensorTlera Sensors;
BME280_t bme280_value;

void setup()
{
    Sensors.initialize_board();
    Sensors.initialize_BME280();
    delay(1000);
    Sensors.callibrate_BME280();

}

void loop()
```

(continues on next page)

(continued from previous page)

```
{
    bme280_value = Sensors.readSensors_BME280();
    Serial.println((String)bme280_value);
    delay(5000);
}
```

5.2 2. Uploading application

To upload the script to the corresponding board, it is requiered to follow the next steps:

5.2.1 2.1. Check hardware setup

The Long-Cricket board has some onboard sensors, which corresponds to BMA400, BME280 and GNSS. On the other hand, you can add external sensors such as ADXL345, L3G4200D y HMC5883L.

Before to connect and attempt to upload the program you should check the connections of external sensors and that they do not generate any conflict with onboard sensors.

5.2.2 2.2. Check software configuration

It is required that sensors have configured correctly, so there should be a corresponding list of libraries and configuration files accompanying them.

Recommended files are considered in *Configuration files*, which provides also a configuration example.

After checked the files your should verify the code in software, for this reason you could follow the next steps:

1. Pressing `F1`, to open the commands pallete.
2. Select the `Arduino:Verify` or via `Ctrl+Alt+R`
3. Verify that it has no errors, showing a message similar as:

```
Loading configuration...
Initializing packages...
Preparing boards...
Verifying...
Sketch uses 121984 bytes (62%) of program storage space. Maximum is 196608 bytes.
Global variables use 8712 bytes (42%) of dynamic memory, leaving 11768 bytes for␣
↪local variables. Maximum is 20480 bytes.
```

5.2.3 2.3. Upload the software

Once verified that there is no errors, you could connect the board to the computer and then:

1. Pressing `F1`, to open the commands pallete.
2. Select the `Arduino:Upload` or via `Ctrl+Alt+U`.
3. The board should be flashed.

CHAPTER

SIX

BUILDING DOCUMENTATION

To build this documentation I follow the excelent guide provided by TartanLlama in Clear, Functional C++ Documentation with Sphinx + Breathe + Doxygen + CMake who also have a Github repository where there is an example to document C++ code.

The process is depicted in the next figure:

Fig. 1: Building documentation process by Luighi Viton, 2020

According to this figure, the documentation is generated employing two main software: Doxygen and Sphinx.

Doxygen in by far the most common alternative when documenting C/C++ code, as it is able to strip out comments and based on them generate a corresponding documentation outputs ina variety of formats such as HTML, LaTeX, XML, etc.

However, to generate Doxygen documentation, all the information should be in the source code, which is unconvinient when you want to create custom pages and instructions. In addition to this, default themes in Doxygen are a bit outdated and appears to be old and not variety user friendly.

Then, to provide more flexibility, there is another tool named Sphinx, which is mainly use to generate documentation for Python projects. Although, it could not manage directly C/C++ code, there is an extension named Breathe which could connect with Doxygen, in the way that the generated output of Doxygen is taken by this extension and based on that, could generate the Sphinx documentation.

In addition to this, there is an online service called Read the Docs, which could publish the documentation based on the Sphinx source files. So, for this build the documentation we have two approaches: the local building documentation and the remote building documentation in the Read the Docs.

6.1 1. Local building process

The local building process makes use of `cmake` to automate the steps required:

```
mkdir build
cd build
cmake ..
make Sphinx
```

Once the building process are completed succesfully, the generated output is place in `build\docs\docs\sphinx`, so you can open the `index.html` file in this folder to explore the documentation.

The control in the generation process is performed by cmake which have a module to generate the Doxygen outputs based on the `Doxyfile.in` file and a custom order for `Sphinx` based on the `conf.py` file and the `*.rst` files created to explain the whole documentation.

6.2 2. Documentation at ReadTheDocs

The documentation at ReadTheDocs is generated in a similar way as in the local building process, however, it differs in the way that the file `conf.py` has the control and is responsible to generate also the Doxygen outputs addition create from them the Sphinx output documentation.

In general, all the changes uploaded to the respository are updated in the documentation as they are connected as a feature provided by the service, so as long as you have updated the documentation in the remote repository, it also will be updated in ReadTheDocs.

6.3 3. Additional resources

- Doxygen commands
- Doxygen directives in Breathe
- Showing code in Sphinx
- reStructuredText in Sphinx
- reStructuredText reference guide

On-premise Read the Docs service:

- Read the Docs Installation
- Docker based Read the Docs

CHAPTER

SEVEN

ABOUT

The library and documentation provided here is part of a project named: *Development of a geolocalization system based on wireless sensors for the Lama Guanicoe biological corridor study and preservation at the Salinas y Aguada Blanca National Reserve, Arequipa*[1], developed by INICTEL-UNI and sponsored by CONCYTEC.

In this case, the library is oriented to help the application software for end devices in the Geolocalization system.

7.1 1. Project description

The project has the aim to develop a geolocalization system based on wireless communication technologies for sensor networks and trilateration algorithms adaptative to metheorological conditions which allow them to have long periods of autonomy, higher than 8 months, required to study the biological corridor and the distribution of the specie at the Salinas y Aguada Blanca National Reserve, Arequipa.

This project is linked to several topics such as IoT, long range wireless communication technologies and energy efficency. This require a embedded software development which is managed, to some extent, by the library described in this documentation.

If you are interested on more details about this project you could contact to the project leader or the research team shown below.

7.2 2. Project team

Team leader Jimi Lezama <jinmilezama@gmail.com> (Embedded systems specialist)

Research team

- Ricardo Yauri <ryauri@inictel-uni.edu.pe> (Embedded HW/SW specialist)
- Oscar Llerena <ollerena@inictel-uni.edu.pe> (Communication systems specialist, Wireless SW developer)
- Ruben Acosta <racosta@inictel-uni.edu.pe> (Embedded HW/SW specialist)
- Luighi Vitón <lviton@inictel-uni.edu.pe> (Embedded SW developer)

LoRaWAN network server administrator Oscar Llerena <ollerena@inictel-uni.edu.pe>

Documentation maintainer Luighi Vitón <lviton@inictel-uni.edu.pe>

[1] Translated from original name: *Desarrollo de un sistema de geolocalización para el estudio del corredor biológico y conservación de la Lama Guanicoe en la Reserva Nacional de Salinas y Aguada Blanca de la región Arequipa basado en sensores inalámbricos.*

INDEX

A

accel_t (*C++ struct*), 15
accel_t::ax (*C++ member*), 16
accel_t::ay (*C++ member*), 16
accel_t::az (*C++ member*), 16
accel_t::bytes (*C++ member*), 16
accel_t::operator String (*C++ function*), 15
accel_t::operator uint8_t* (*C++ function*), 15
ADXL345_t (*C++ struct*), 23
ADXL345_t::bytes (*C++ member*), 23
ADXL345_t::operator String (*C++ function*), 23
ADXL345_t::operator uint8_t* (*C++ function*), 23
ADXL345_t::status (*C++ member*), 23
ADXL345_t::values (*C++ member*), 23

B

batt_t (*C++ struct*), 20
batt_t::bytes (*C++ member*), 20
batt_t::operator int16_t (*C++ function*), 20
batt_t::operator String (*C++ function*), 20
batt_t::operator uint8_t* (*C++ function*), 20
batt_t::STM32L0Temp (*C++ member*), 20
batt_t::vbat (*C++ member*), 20
batt_t::vbus (*C++ member*), 20
batt_t::vdda (*C++ member*), 20
BMA400_t (*C++ struct*), 21
BMA400_t::offset (*C++ member*), 22
BMA400_t::resolution (*C++ member*), 22
BMA400_t::status (*C++ member*), 22
BMA400_t::values (*C++ member*), 22
BME280_t (*C++ struct*), 22
BME280_t::bytes (*C++ member*), 22
BME280_t::humidity (*C++ member*), 22
BME280_t::operator String (*C++ function*), 22
BME280_t::operator uint8_t* (*C++ function*), 22
BME280_t::pressure (*C++ member*), 22
BME280_t::status (*C++ member*), 22
BME280_t::temperature (*C++ member*), 22
board_t (*C++ enum*), 12
board_t::CRICKET (*C++ enumerator*), 12
board_t::GNAT (*C++ enumerator*), 12
board_t::GRASSHOPPER (*C++ enumerator*), 12
BOARD_TYPE (*C macro*), 11

C

CRI (*C macro*), 12

D

DEBUG (*C macro*), 11
DEBUG_ADXL345 (*C macro*), 12
DEBUG_BMA400 (*C macro*), 12
DEBUG_BME280 (*C macro*), 12
DEBUG_BOARD (*C macro*), 12
DEBUG_GNSS (*C macro*), 12
DEBUG_HMC5883L (*C macro*), 12
DEBUG_L3G4200D (*C macro*), 12

G

GATEWAY_TEKTELIC (*C macro*), 13
GNAT (*C macro*), 13
GNSS_t (*C++ struct*), 21
GNSS_t::altitude (*C++ member*), 21
GNSS_t::bytes (*C++ member*), 21
GNSS_t::isSpatial (*C++ member*), 21
GNSS_t::latitude (*C++ member*), 21
GNSS_t::location (*C++ member*), 21
GNSS_t::longitude (*C++ member*), 21
GNSS_t::operator uint8_t* (*C++ function*), 21
GRASS (*C macro*), 13
gyro_t (*C++ struct*), 16
gyro_t::avx (*C++ member*), 16
gyro_t::avy (*C++ member*), 16
gyro_t::avz (*C++ member*), 16
gyro_t::bytes (*C++ member*), 16
gyro_t::operator String (*C++ function*), 16
gyro_t::operator uint8_t* (*C++ function*), 16

H

HMC5883L_t (*C++ struct*), 24
HMC5883L_t::bytes (*C++ member*), 25

HMC5883L_t::heading (C++ member), 25
HMC5883L_t::operator String (C++ function), 24
HMC5883L_t::operator uint8_t* (C++ function), 24
HMC5883L_t::status (C++ member), 25
HMC5883L_t::values (C++ member), 25
hum_t (C++ struct), 18
hum_t::bytes (C++ member), 19
hum_t::compensated (C++ member), 19
hum_t::hum_relative (C++ member), 19
hum_t::operator int16_t (C++ function), 18
hum_t::operator String (C++ function), 18
hum_t::operator uint8_t* (C++ function), 18
hum_t::raw (C++ member), 19

L

L3G4200D_t (C++ struct), 23
L3G4200D_t::bytes (C++ member), 24
L3G4200D_t::operator String (C++ function), 24
L3G4200D_t::operator uint8_t* (C++ function), 24
L3G4200D_t::status (C++ member), 24
L3G4200D_t::values (C++ member), 24
LED_BOARD (C macro), 12
LORA_SETTINGS (C macro), 14
LORA_SETTINGS_CRI (C macro), 14
LoRaSettings (C++ struct), 13
LoRaSettings::appEui (C++ member), 14
LoRaSettings::appKey (C++ member), 14
LoRaSettings::devEui (C++ member), 14
LoRaSettings::id (C++ member), 14

M

mag_t (C++ struct), 17
mag_t::bytes (C++ member), 17
mag_t::mx (C++ member), 17
mag_t::my (C++ member), 17
mag_t::mz (C++ member), 17
mag_t::operator String (C++ function), 17
mag_t::operator uint8_t* (C++ function), 17
MYAPPKEY (C macro), 13
MYDEVICE (C macro), 13

P

press_t (C++ struct), 19
press_t::altitude (C++ member), 19
press_t::altitude_mts (C++ member), 19
press_t::bytes (C++ member), 19
press_t::compensated (C++ member), 19
press_t::operator int16_t (C++ function), 19
press_t::operator String (C++ function), 19
press_t::operator uint8_t* (C++ function), 19
press_t::press_mbar (C++ member), 19
press_t::raw (C++ member), 19

S

SensorsTlera (C++ class), 25
SensorsTlera::callibrate_BMA400 (C++ function), 26
SensorsTlera::callibrate_BME280 (C++ function), 26
SensorsTlera::getUID (C++ function), 25
SensorsTlera::GNSS_isSpatial (C++ function), 27
SensorsTlera::initialize_ADXL345 (C++ function), 25
SensorsTlera::initialize_BMA400 (C++ function), 25
SensorsTlera::initialize_BME280 (C++ function), 25
SensorsTlera::initialize_board (C++ function), 25
SensorsTlera::initialize_GNSS (C++ function), 25
SensorsTlera::initialize_HMC5883L (C++ function), 26
SensorsTlera::initialize_L3G4200D (C++ function), 25
SensorsTlera::readSensors_ADXL345 (C++ function), 26
SensorsTlera::readSensors_batmon (C++ function), 26
SensorsTlera::readSensors_batmon_average (C++ function), 26
SensorsTlera::readSensors_BME280 (C++ function), 26
SensorsTlera::readSensors_GNSS (C++ function), 27
SensorsTlera::readSensors_HMC5883L (C++ function), 27
SensorsTlera::readSensors_L3G4200D (C++ function), 26
SensorsTlera::scanI2CDevices (C++ function), 27
SensorsTlera::SensorsTlera (C++ function), 25

T

temp_t (C++ struct), 17
temp_t::bytes (C++ member), 18
temp_t::compensated (C++ member), 18
temp_t::operator int16_t (C++ function), 17
temp_t::operator String (C++ function), 18
temp_t::operator uint8_t* (C++ function), 17

temp_t::raw (*C++ member*), 18
temp_t::temp_C (*C++ member*), 18
temp_t::temp_F (*C++ member*), 18

U

USER_BUTTON (*C macro*), 12

V

VBAT_CTRL (*C macro*), 12
VBAT_MON (*C macro*), 12

W

WITH_ADXL345 (*C macro*), 11
WITH_BMA400 (*C macro*), 11
WITH_BME280 (*C macro*), 11
WITH_GNSS (*C macro*), 11
WITH_HMC5883L (*C macro*), 11
WITH_L3G4200D (*C macro*), 11